

AFRL-IF-RS-TR-2006-178
In-House Interim Report
May 2006



JOINT BATTLESPACE INFOSPHERE (JBI): INFORMATION MANAGEMENT IN A NETCENTRIC ENVIRONMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

**AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK**

STINFO FINAL REPORT

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2006-178 has been reviewed and is approved for publication

APPROVED: /s/

STEVEN D. FARR
Chief, Systems Information and Interoperability Branch
Information Systems Division

FOR THE DIRECTOR: /s/

JAMES W. CUSACK
Chief, Information Systems Division
Information Directorate

REPORT DOCUMENTATION PAGE			<i>Form Approved</i> <i>OMB No. 074-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE MAY 2006	3. REPORT TYPE AND DATES COVERED In-House Interim Mar 01 – Apr 06	
4. TITLE AND SUBTITLE JOINT BATTLESPACE INFOSPHERE (JBI): INFORMATION MANAGEMENT IN A NETCENTRIC ENVIRONMENT			5. FUNDING NUMBERS C - N/A PE - 63789F PR - JBIH0001 TA - N/A WU - N/A	
6. AUTHOR(S) Mark Linderman, Vaughn T. Combs, Robert G. Hillman, Mike T. Muccio, Ryan W. McKeel				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/IFSE 525 Brooks Road Rome New York 13441-4505			8. PERFORMING ORGANIZATION REPORT NUMBER N/A	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/IFSE 525 Brooks Road Rome New York 13441-4505			10. SPONSORING / MONITORING AGENCY REPORT NUMBER AFRL-IF-RS-TR-2006-178	
11. SUPPLEMENTARY NOTES AFRL Project Engineer: Robert G. Hillman/IFSE/(315) 330-4961/ Robert.Hillman@rl.af.mil				
12a. DISTRIBUTION / AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.				12b. DISTRIBUTION CODE
13. ABSTRACT (Maximum 200 Words) The Joint BattleSpace Infosphere (JBI) is an information management concept developed by the Air Force Scientific Advisory Board (SAB) to address information management challenges in a military environment. Throughout the ensuing years, the Systems and Information Interoperability Branch (AFRL/IFSE) has performed research in information management that has further refined the original JBI concepts and has demonstrated how those concepts may be applied to achieve system interoperability. A JBI comprises many diverse applications (called clients) and a set of core services that enable the dissemination, persistence and control of information being shared among the applications. One key core service provided to disseminate information is publish and subscribe (Pub/Sub). In this interim report, we describe the technical details of the design and implementation of two different JBI Reference Implementations (RIs) core services technology on which both are based on publish/subscribe, but took different approaches, a Jini-based and a J2EE-based JBI RIs.				
14. SUBJECT TERMS Joint BattleSpace Infosphere, JBI, J2EE, Jini, XML, Publish and Subscribe				15. NUMBER OF PAGES 37
				16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	

TABLE OF CONTENTS

1.	EXECUTIVE SUMMARY	1
2.	OVERVIEW OF THE JBI RI PROJECT	2
3.	WHY PUBLISH AND SUBSCRIBE?.....	3
4.	JINI-BASED PUBLISH AND SUBSCRIBE CORE SERVICES	4
4.1.	THE ROLE OF METADATA	5
4.2.	WHAT IS JINI?	6
4.3.	BROKERING.....	7
4.3.1.	USING JINI WITHIN THE PUBLISH AND SUBSCRIBE INFRASTRUCTURE.....	7
4.3.2.	USING XML WITHIN PUBLISH AND SUBSCRIBE INFRASTRUCTURE.....	8
4.3.3.	INTERACTIONS WITHIN THE PUB/SUB SYSTEM	11
5.	EXPLOITATION OF THE INFORMATION SPACE	14
5.1.	AIRBORNE EXPERIMENT.....	14
5.2.	COABS GRID INTEGRATION	15
5.3.	JOINT EXPERIMENTATION	16
6.	JAVA 2 ENTERPRISE EDITION (J2EE)-BASED JBI REFERENCE IMPLEMENTATION	16
6.1.	CLIENT APPLICATION	16
6.2.	THE INFORMATION OBJECT	16
6.3.	THE COMMON API (CAPI)	17
6.4.	VERSION 1.2 ARCHITECTURAL DESIGN.....	18
7.	FUTURE DIRECTIONS.....	21
7.1.	DISCOVERY	22
7.2.	POLICY REPRESENTATION AND ENFORCEMENT	22
7.3.	INSTRUMENTATION AND CONTROL	24
7.4.	INFORMATION TRANSFORMATION.....	25
7.5.	JBI AND NET-CENTRIC SYNERGY	26
7.6.	COMMUNITY OF INTEREST	26
7.7.	AIR FORCE C2 CONSTELLATION	26
7.8.	GLOBAL INFORMATION GRID ENTERPRISE SERVICES.....	27
7.9.	NET-CENTRIC ENTERPRISE SERVICES	27
7.10.	JBI AND GIG ENTERPRISE SERVICES	28
7.11.	JBI AND GIG NET-CENTRIC SERVICES.....	29
8.	CONCLUSION	30
9.	REFERENCES.....	31

List of Figures

FIGURE 1: EXAMPLE OF A METADATA FOR INTELLIGENCE IMAGERY	5
FIGURE 2: INVARIANT METADATA USED TO POPULATE ENTRY CLASSES	8
FIGURE 3: SUBSCRIPTION TEMPLATE FOR INVARIANT METADATA	9
FIGURE 4: AN EXAMPLE OF AN XML PAYLOAD FOR A TARGET LIST	10
FIGURE 5: INTERACTIONS WITHIN THE JBI JINI-BASED PUBLISH AND SUBSCRIPTION SERVICES	12
FIGURE 6: THE PLUGGABLE COMMON API	18
FIGURE 7: JBI CORE SERVICES.....	20
FIGURE 8: JBI INSTRUMENTATION VISUALIZATION CLIENT	24
FIGURE 9: AN EXAMPLE OF A SERVICE	28
FIGURE 10: JBI ENABLED INFORMATION SPACE FACILITATE EDGE USER INTERACTION IN CONJUNCTION WITH GIG ENTERPRISE SERVICES.....	28
FIGURE 11: INITIAL EVALUATION OF GIG ES INTERSECTION.....	30

1. Executive Summary

This interim technical report provides the details concerning the result of the research and the development performed for the Joint Battlespace Infosphere (JBI) program, an information management in a net-centric environment. The concept of the JBI was first created in 1999 by the US Air Force (USAF) Scientific Advisory Board (SAB) [1] [2] [3] to address a clear lack of information engineering and interoperability within and among existing, traditionally stove-piped fielded capability. Systems interoperability continues to be one of the key challenges facing the DoD as new capability is developed and fielded while maintaining operation with existing legacy systems. Soon after the SAB published their reports, engineers, and scientists in what is now the Systems and Information Interoperability Branch (AFRL/IFSE) began researching the technologies and techniques that one could exploit to solve this rather unique set of information management challenges. As a product of this research several in-house Reference Implementations (RIs) were developed in order to prove efficacy of approach while providing developers and external researchers with a substrate upon which they could build their applications and research solutions.

Throughout the ensuing years, the Systems and Information Interoperability branch has performed research in information management that has further refined the original JBI concepts and has demonstrated how those concepts may be applied to achieve system interoperability. JBI information services allow for mediated, loosely-coupled access to information among systems and system components using publish and subscribe paradigm. In addition, a query capability is provided for access to archived information. All data exchanged is treated as managed information. Each object published contains associated metadata that is used to broker for information object instances between producers and consumers. The consumers of information provide predicates or constraints over the metadata which are subsequently used to decide what information is appropriate for dissemination to the requesting clients.

The initial JBI RI was built on a Jini-based publish/subscribe core services technology. As the technology evolved over the time, the latest released RI is built on a new techniques that is still based on publish/subscribe paradigm but used Java2 Enterprise Edition (J2EE) technologies for interoperability among systems in a more standardized and efficient way. In this report, a detailed description of the Publish/Subscribe system design and implementation will be given that describes how and where Java, Jini, and XML technologies were used to describe information objects, match subscribers to appropriate dissemination nodes, and disseminate information objects to subscribing clients followed by a details description of the current J2EE RI (version 1.2) platform architecture design and implementation.

More recently, the Defense Information Systems Agency (DISA) has sought to provide an architectural design and implementation of a collection of underlying services that would be available to edge user client applications and systems. These services allow clients to share information intelligently from anywhere within the network environment

using a post, discover and pull methodology. These Net-Centric Enterprise Services (NCES) are being developed as a Service-Oriented Architecture (SOA) that uses the latest industry standards such as XML and Web Services. In this report, we also discuss how the JBI core capability operating within the Global Information Grid (GIG) will utilize the NCES services to provide an information space capable of integrating, fusing, aggregating and intelligently disseminating information to war-fighter business processes.

2. Overview of the JBI RI Project

The Joint Battlespace Infosphere (JBI) traces its origins to the USAF Scientific Advisory Board study that was completed in 1999 [3]. In this document, they describe the vision of a combat information management system that would integrate information from a wide variety of sources, aggregate that information and then disseminate the information in the proper format and level of detail to appropriate consumers. In short, the JBI will provide the warfighter, subject to policy, access to the right information in the right format and at the right time.

An additional requirement of the JBI is to support a level of information interoperability that is absent in currently fielded systems. These systems tend to use point-to-point interfaces in order to link together traditionally stove-piped systems. To achieve the JBI interoperability goal, systems should be treated as a collection of loosely coupled components that issue requests for information that satisfy their individual requirements. This levies an additional derived requirement on the JBI and all of its clients that all information must be characterized before it is submitted to the Infosphere. It is in fact this metadata that plays a key role in the effective management and dissemination of information between producers and consumers. Information published to the Infosphere is often referred to as an Information Object (IO) or a Managed Information Object (MIO).

In order to consume information objects, clients may use one of two fundamental mechanisms: subscribe or query. Query is used to retrieve information that has been published to the JBI in the past, while subscription is used to receive information that may be published in the future. In both cases, a constraint or predicate is supplied over the metadata that is published with each MIO in order to assure that clients receive only the information that they are interested in.

The JBI provides developers with a capability for embedding legacy business logic or other transformation code as lightweight decision logic whose lifecycle is controlled by the underlying Infosphere. This decision logic, referred to as a *fuselet*, is completely maintained and managed by the JBI platform and may potentially act on behalf of a number of client applications.

In addition, the Infosphere provides a capability to dynamically alter the information space and the configuration of underlying services as operational units enter or leave the virtual infospace. Each deployed unit defines its capabilities, support needs, and information interface (subscription and publish descriptions) through the use of *Force Templates*. Force templates define the “electronic handshake” between the JBI and subordinate units, and their use lets units be quickly added to the JBI with little or no manual reconfiguration required. This capability allows individual units to identify their capabilities and needs via a collection of *Force Templates*.

Finally, the Infosphere may be dynamically administered by information Management Staff (IMS) personnel. It is their responsibility to enforce the Commander’s intent represented as policy (security, QoS, etc.), which is enforced by the platform.

3. Why Publish and Subscribe?

As mentioned in the Air Force Scientific Advisory Board (SAB) report, the publish/subscribe mechanism was the best choice to promote interoperability and reduce cost. Why?

When assembling a coalition of allies, as is often the case today, commanders must rely on systems that have the capability to characterize information completely enough so that the right information gets to the right coalition partners at the right time. A publish and subscribe paradigm may be used to richly characterize information object using metadata descriptions. Typically the producer of the information object creates a metadata descriptor associated with a distinct information object instance and then publishes it with the object. The subscriber or consumer of the information submits a subscription predicate defined over the metadata associated with information objects of that type. The underlying core services may then use this information to broker for the delivery of information objects between the producers of the information and the consumers of the information. Such a system allows for disparate military systems to share and disseminate information with no consideration for individual system interfaces.

In addition, most military system architects are coming to the conclusion that building systems that are tightly tied to the most recent substrate technologies makes these systems resistant to change. It is precisely this limitation that has led the JBI group to work toward a set of Common Core Service Application Programming Interfaces APIs (these APIs are under development). The specification of the Common API is an ongoing DoD community processes which is making considerable progress. This approach would allow developers to design and implement client applications that need not consider what underlying substrate is being used for a particular JBI platform implementation. This notion alone is powerful in the context of military system implementations. Legacy applications that are built on JBI publish, subscribe, and query capabilities would no longer need to reinvest considerable resources to take advantage of the most recent shifts in technology. JBI platform implementers would implement the client API using the new technologies and existing client applications would run with, conceivably, no alteration.

4. Jini-Based Publish and Subscribe Core Services

A JBI comprises many diverse applications (called clients) and a set of core services that enable the dissemination, persistence and control of information being shared among the applications. One of the key core services provided to disseminate information is publish and subscribe (pub/sub). Pub/sub was chosen because it encourages very loose coupling between applications; it is believed that traditional military systems are too tightly coupled leading to interoperability and extensibility challenges.

However, it was thought that the traditional ‘channel-based’ pub/sub mechanisms did not provide enough fine-grain control to ensure that subscribers receive only appropriate information. Indeed, while a hierarchical tree of channels is useful to capture the essence of conceptual refinement, it is insufficient to deal with vast amounts of data – all of an identical ‘type’ or channel. Previous to JMS 1.0.2, [4] the only solution was to create sub-channels to segregate information, but it was thought that this was too static in nature. For example, partitioning channels by geographical extent works if the regions of interest are static, but if the data represent the forward line of battle, this is constantly shifting. The channel hierarchy cannot be redefined in real time.

To address this issue and to facilitate interoperability, it was decided that information would be structured, and that subscribers would indicate their information needs with predicates over the structured information. However, not all information is naturally structured or likely to be used in predicates (e.g. images), so it was decided that the information would be a) typed, b) have a payload that may contain anything, and c) have structured metadata describing the payload. These three elements combine to form the information object. All applications using the JBI to exchange information must provide that information as a typed information objects.

The pub/sub infrastructure is responsible for applying subscriber predicates to the metadata of published information objects. If the metadata of an information object satisfies a predicate, the information object is forwarded to the subscriber. This Jini-based mechanism is mainly used to broker publishers and subscribers efficiently and push information objects from publishers to subscribers. Efficient brokering is based upon publishers registering before starting to publish. During the registration process, the publisher indicates its *invariant metadata*. Invariant metadata are elements of the information object metadata that will be constant in all subsequently published information objects. This information permits an efficient first-level matching of subscribers and publishers. Subsequent matching, while more sophisticated, incurs a runtime penalty for subscriber predicate evaluation for each potentially matching subscriber each time an information object is published.

This implementation uses XML and XML Schema to represent metadata. Publishers provide invariant metadata as an XML document at time of registration. The underlying core service uses the schema to identify the invariant portions of the metadata to be used for registration with Jini’s broker (lookup service). In addition to brokering based on the invariant aspects of the metadata, the current implementation allows for subscription using an arbitrary XPath or XQL predicate. This capability enhances Jini’s equality based matching capabilities with support for inequality, AND, OR, NOT, etc. In addition, when

the published information object payload happens to be in XML format (not a requirement), the pub/sub service permits arbitrary content-based filtering as part of the subscriber API. In addition to disseminating published information objects directly into the address space of subscribing applications, the Pub/Sub system also allows for indirect dissemination via email. Currently, the Pub/Sub capability is being used by over 30 DoD contractors and is in active use for internal research and development projects

The following sections give a brief description of how the JBI Jini-Based Publish and Subscribe core services work. This implementation provides a very convenient set of API method signatures for use by JBI publishing and subscribing client applications. While this implementation does not implement the aforementioned Common API, it does provide an API for publication and subscription that hides all of the underlying Jini specifics from the client developer. The Common API will be implemented on top of the current implementation as a set of veneer classes as the API becomes more mature (see the J2EE-based JBI Core Services Section).

4.1. The Role of Metadata

In a JBI, all information objects are characterized via metadata. In this implementation we have chosen to describe the metadata using XML [5]. Corresponding to each information object type is an XML Schema [6] that describes the structure of its metadata. The metadata is used by the underlying core services to match subscribers to information objects that are of interest. Figure 1 below shows a simple example of metadata instance that describes a particular information object.

```
<?xml version="1.0" encoding="UTF-8"?>
<metadata xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Intel_Imagery.xsd">
  <mil.af.rl.jbi.Intel_Imagery/>
  <RequiredMetadata>
    <Type>Imagery</Type>
    <JBIIIdentifier>JBI000023</JBIIIdentifier>
    <publisher>418th</publisher>
    <keywords>Intel</keywords>
    <language>EN</language>
  </RequiredMetadata>
  <ImageDescriptor>
    <ImageType>IR</ImageType>
    <Area>Kabul</Area>
    <LocationCoord>
      <lat>33.34</lat>
      <latord>N</latord>
      <long>69.98</long>
      <longord>E</longord>
    </LocationCoord>
  </ImageDescriptor>
</metadata>
```

Figure 1: Example of a Metadata for Intelligence imagery

In the above XML example, the information object payload happens to be intelligence imagery in the Kabul area. The *information object type name* is defined to be `mil.af.rl.jbi.Intel_Imagery`. For each information object in the JBI there exists a unique metadata schema that describes the structure of the metadata for the type. The schema is used by the developer of the publishing application to generate valid metadata describing the object being published. The schema is similarly used by the subscribing application developer to create a predicates over the metadata that describe the subscriber's information requirements.

In our implementation we distinguish between the *invariant and variant* aspects of metadata. In the example above we may wish to consider the attribute *"RequiredMetadata"* subelement to be invariant. In other words, for this session, we could reasonably expect that the type, JBIIdentifier, publisher, keywords and language would not change for each of the images published. In contrast, we could consider all of the *"ImageDescriptor"* information to be a variant sub-element of the metadata. For example, we would expect that, in general, the location of the target depicted in the image published would change with each published image. While this would seem to be an unnecessary distinction, we will describe how the identification of the invariant and variant portions of the metadata may be used to increase the efficiency of the brokering mechanisms used in this implementation. Currently we annotate sub-elements within the XML Schema with comments denoting whether a sub-element is variant. In future releases we will allow the user to specify this using XML attributes.

Work has begun to eliminate the additional specification of a subscription template regarding the invariant aspects of the metadata. The schema would then also be used to parse the XPath predicate for equality based expressions. This information would be used in the creation and population of the necessary underlying Entry objects.

4.2. What Is JINI?

Jini [7] is a freely available service oriented architecture that was originally designed as a distributed computing environment to provide networked devices with a network level plug-and-play capability. The architecture specifies how clients and services find each other within a "community". The service implementers supply Jini and, in turn, clients with a portable Java-based object (proxy) that can be used to allow the client to access and interact with the service. While, in most cases, this proxy uses Java Remote Method Invocation (RMI), any network technology may be used (i.e. CORBA, SOAP, etc.).

When a service joins a network of Jini enabled services, it must advertise its capabilities by publishing the proxy implementing the services' exported API. The proxy is registered with Jini's Lookup Service (LUS). A client application may then request the proxy for a specific service. A typical example may be to request the proxy for a "Printer Service". This would probably not be desirable, since you would receive the proxies for all printers that are currently registered. You may, alternatively, wish to interact with a printer within

the same building that is capable of printing in color. Jini allows the service to further characterize itself using classes that describe its attribute value pairs. The Jini LUS would then allow the client application to use the additional attributes to specify a limited filtering capability. Currently the Jini LUS only allows for equality based matching with wildcarding. In our previous example one could then easily request the proxies to all printer services that are capable of printing in color and/or all printer services within a specific building or, if desired, all printer services. When the client application receives the service's published proxy object, it will download any code it needs in order to interact with the service.

Jini supports a rather robust set of protocols to enable the spontaneous discovery of services as they enter (and leave) a community. Should a client wish to use a specific service and it is not currently registered within the community, the client may choose to be notified when one becomes available. Analogously, the client may also choose to be notified when a service leaves the community. Additional Jini lookup services may be instantiated within the community to support a collection of lookup services that are tied to specific groups or simply to provide for a certain level of fault tolerance. In this case the clients and services within the community are notified of the new LUS and are afforded the opportunity to register with it.

4.3. *Brokering*

One of the most essential capabilities within a publish/subscribe system is a well-designed brokering capability. The fundamental responsibility of the broker is to the match information objects produced by a publisher with appropriate subscribers. This section describes how the Jini-based Publish and Subscribe core services leverage the brokering capability implemented within the Jini Lookup Service with XML based technologies to provide efficient matching and information object dissemination services.

4.3.1. *Using Jini within the Publish and Subscribe Infrastructure*

In a previous section, we touched briefly on Jini's ability to characterize services based on attributes. Jini allows a service to register a proxy with its LookUp Service (LUS). The proxy is ultimately used by remote clients to interact with the service. In addition, the service may associate a number of classes that implement the *Entry* interface with the proxy. These Entry class instances contain simple attribute/value pairs that are used by the LUS for the equality-based matching capabilities described above. If a successful match occurs, the proxy is then handed off to the client and the client may then interact directly with the service.

We use this capability to enable a first level of matching between published objects and subscribers. As mentioned above, we distinguish between the invariant and variant aspects of metadata. The invariant metadata is used to specify attribute value pairs that will be registered with the Jini LUS. Specifically, our implementation uses the schema for the information object metadata and the metadata instance provided by the publishing

application to auto generate, instantiate and populate the Entry classes that we use to register with the Jini LUS. The associated “proxy” that is registered is actually an exported API that will be used by the JBI’s subscription side classes to register with appropriate dissemination nodes (delegate model) or publishing applications (peer-to-peer model).

The subscribing application, using the JBI subscription API, may then register a subscription template and predicate. The subscription template (an XML document) is used to specify the first level filtering that we would like to apply over the invariant attributes. Again, unknown to the subscribing application, we generate, instantiate, and populate all Entry classes needed by the underlying Jini substrate and use the classes to register our “subscription template”. We will discuss both publish and subscribe side interactions in greater detail in a later section.

4.3.2. Using XML within Publish and Subscribe Infrastructure

Previously we have mentioned that this implementation treats variant and invariant metadata differently. Using our previous example the publishing application first would register to publish providing an XML document that specifies information object type and the invariant element values that are appropriate for the registered sequence of published objects.

```
<?xml version="1.0" encoding="UTF-8"?>
<metadata xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Intel_Imagery.xsd">
  <mil.af.rl.jbi.Intel_Imagery/>
  <RequiredMetadata>
    <Type>Imagery</Type>
    <JBIIIdentifier>JBI000023</JBIIIdentifier>
    <publisher>418th</publisher>
    <keywords>Intel</keywords>
    <language>EN</language>
  </RequiredMetadata>
</metadata>
```

Figure 2: Invariant metadata used to populate Entry classes

The information shown in Figure 2 would be used to populate the generated Entry classes and subsequently register with the Jini LUS. Upon publication, a full complement of metadata would be provided (both variant and invariant) not unlike the example shown in section 2.1. The subscribing client application may provide two pieces of information. First, a subscription template must be defined for the *invariant portions* of the metadata (Figure 3). This template is essentially an XML document that describes the equality

based matching with wildcarding semantics that would be provided as our first level match between candidate publication and subscription.

```
<?xml version="1.0" encoding="UTF-8"?>
<metadata xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Intel_Imagery.xsd">
  <mil.af.rl.jbi.Intel_Imagery/>
  <RequiredMetadata>
    <Type>Imagery</Type>
    <JBIIIdentifier/>
    <publisher>418th</publisher>
    <keywords/>
    <language>EN</language>
  </RequiredMetadata>
</metadata>
```

Figure 3: Subscription template for invariant metadata

Notice that, in the above example, the subscriber has decided that she is interested in all Imagery published by the 418th that is annotated in English. In other words, the client has wildcarded the JBIIIdentifier and keywords fields.

While this first level of matching is very powerful it is inadequate given the matching semantics that are necessary within a JBI (i.e. inequality, AND, OR, NOT, etc.). To support the appropriate level of matching the Pub/Sub core services use XPath or XQL technologies to do additional predicate testing in the dissemination nodes and peer publishers (default implementation uses XPath).

To recap, the first level match is made and, if successful, a registration process occurs that allows the subscriber to register an optional XPath expression. The expression will then be used as a *predicate* that will be applied to the metadata that is published with each of the information objects of interest. An example of the instance metadata may be found in Figure 1. If the metadata describes an object that the subscriber wants then it will be disseminated.

The following is a simple but very powerful XPath expression that is consistent with our example:

```
*[(/metadata/ImageDescriptor/LocationCoord/lat>33.2) and
(/metadata/ImageDescriptor/LocationCoord/lat<35.0) and
(/metadata/ImageDescriptor/LocationCoord/latord='N') and
(/metadata/ImageDescriptor/LocationCoord/long>65.7) and
(/metadata/ImageDescriptor/LocationCoord/long<70.0) and
(/metadata/ImageDescriptor/LocationCoord/longord='E')]
```

The expression defines a rectangular region in and around Kabul, Afghanistan. If this predicate were specified as part of the subscription then only the intelligence images that are of targets within the defined rectangular region would be disseminated to the subscriber.

Additionally, the subscriber may optionally provide an XPath *filter* that would be applied to the payload of the published information object. If the payload is in XML format (certainly not a requirement) the dissemination nodes will apply the filter to the payload and disseminate only that part of the payload that is of interest. In contrast to our previous example the payload depicted in Figure 4 contains a target list. Each target is characterized by a descriptor that is a sub-element within the information object payload (XML in this case). The subscriber may provide a content-based filter to return only target descriptors that are in a specific rectangular geographical region as follows:

```
//descriptor[(((./position/lat>'33.2') and (./position/lat<'35.0')) and (./position/ladir='N'))
and (((./position/long<'70.0') and (./position/long>'67.8')) and
(./position/lodir='E'))]
```

```
<targets>
  <descriptor>
    <name>ALPHA50</name>
    <type>F15E</type>
    <position>
      <lat>35.217</lat>
      <ladir>N</ladir>
      <long>62.267</long>
      <lodir>E</lodir>
    </position>
  </descriptor>
  <descriptor>
    <name>BRAVO51</name>
    <type>A10A</type>
    <position>
      <lat>34.817</lat>
      <ladir>N</ladir>
      <long>67.817</long>
      <lodir>E</lodir>
    </position>
  </descriptor>
  ...
</targets>
```

Figure 4: An Example of an XML payload for a target list

Again, the distinction between the XPath predicate and the XPath filter is that the *predicate is only applied to the metadata* characterizing the published information object. The *filter is applied to the payload* (if in XML format) to determine what portions of the payload should be disseminated to the subscriber.

4.3.3. *Interactions within the Pub/Sub System*

This section describes, in greater detail, the interactions that take place within the JBI Jini-Based Publish and Subscription Services.

Our implementation supports both peer-to-peer and delegate implementations. These implementations may co-exist with one another. The client applications interact with the system via three adapter classes. The subscribing application uses a Subscriber Adapter (SA) while a publishing application may use a Publisher Adapter (PA) for peer-to-peer publication or a Publisher Adapter Light (PAL) class when using delegates for dissemination.

Using Figure 5 below, we will quickly walk through publish and subscribe sequences. We will go into greater detail later in this section. In steps 1 through 4 the publishing client registers its intention to publish with the JBI. It is at this point that the client provides the XML Schema representing the structure of the metadata used to characterize the information object type and an XML document specifying the invariant metadata element values as discussed in the last section. In steps 5 through 8 the subscribing client registers its subscription with the JBI. In this case the subscription is first matched to a candidate dissemination node using the XML document subscription template. The adapter then registers with the appropriate dissemination node(s) optionally providing an additional predicate and/or filter that may be applied to the published instance metadata and payload.

The following discussions describe the use and, at a high level, the interactions specifically when a publishing application uses a Publisher Adapter Delegate (PAD).

A PAD is a dissemination node that handles all publishing related processing on behalf of a publishing application. Currently a PAD is characterized simply via two arbitrary strings that may be used by a publishing application to broker for a distinct delegate. Any number of delegates may be started within the environment and they may be characterized in any way that is deemed appropriate.

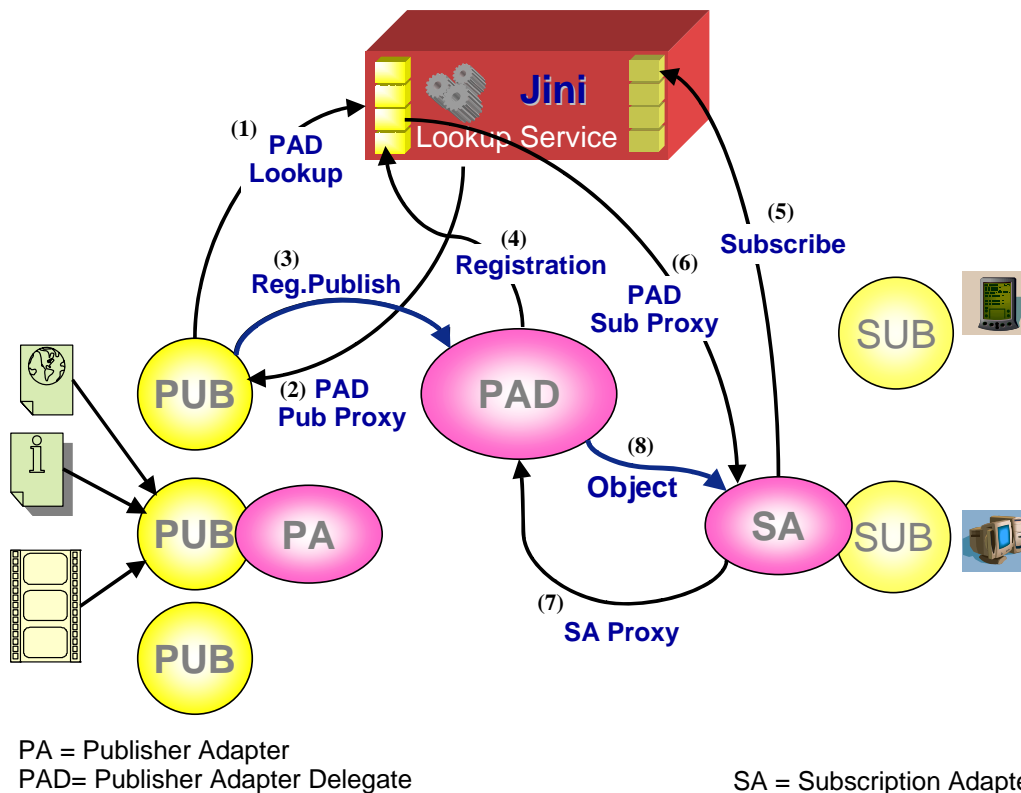


Figure 5: Interactions within the JBI Jini-Based Publish and Subscription Services

In (1) the publishing application uses the *connectToDelegate* method within the Publisher Adapter Light API to connect to a distinct PAD. This method will return a unique identifier that may be used in subsequent PAL API calls to interact with a distinct delegate. This technique is employed to support the use of multiple delegates within one publishing application.

A great deal of care was taken in creation and management of leases between the PAL implementation classes and a matched PAD. Since the PAD consumes resources on behalf of each of its matched publishers (maintaining publication queues, dissemination threads, registrations with LUS, etc.) it was important to implement a lease between the PAL and matched PAD. The lease times used are all configurable via user properties that may be specified at startup. Should a publishing application lose contact with a PAD or die the PAD will clean up after the publisher in an appropriate manner.

In (2) a proxy is returned that will be used by the implementing PAL classes for all interactions with the matched delegate. The proxy itself is never used by the publishing application. Instead the adapter implementation classes use the proxy to interact with the PAD on behalf of the client application. Moreover, the using application is never aware of the fact that it is using Jini.

In (3) the user invokes the *registerPublisher* method within the PAL API to register a publishable object with the JBI Pub/Sub. The client application provides the information object metadata's XML Schema and the invariant information object metadata attribute values in XML format. The information is then used by the underlying PAL implementation classes to generate the necessary Jini Entry classes. Subsequently, the classes are used for registration of the invariant metadata attribute values with the Lookup Service. The instantiated Entry class instances are sent to the PAD as part of the registration process. The registerPublisher method returns another UID that identifies this particular publishable information object registration.

While the Entry class code generation software utilizes the XML Schema to define the structure of the classes, for the most part, the code is class-based and hardcoded. In subsequent releases the code generation will be accomplished using the XML Schema and an associated XSL stylesheet. This approach will allow us to flexibly change how the Entry class code is generated from the XML Schema should the overall metadata structure evolve over time. Any overall changes in metadata representation and structure could then be handled without having to alter the underlying implementation code.

In (4) the PAD registers the publishable object's invariant metadata with the Jini LUS for brokering. At this point the publishing application may use one of the *publish* method signatures supported by the PAL API to publish information objects and associated metadata. To recap, the publishing application uses variations of three very simple method signatures *connectToDelegate*, *registerPublisher*, and *publish*.

In (5) we begin to look at a subscriber's use of the Subscriber Adapter API. The subscribing application simply uses one of the SA API *registerSubscriber* methods to register a subscription. The method allows for the specification of an XML document that is the metadata template. The template is essentially used to specify the equality based matching criteria over the invariant portions of the metadata. Entry classes that reflect this information are then created, instantiated, populated and used to register this first level of matching criteria with the Jini LUS. It is at this point that the XPath predicate may also be specified as an argument that will eventually be applied by the PAD over the metadata published with a specific information object instance. In addition, it is at this point that a subscriber may specify a content-based filter in XPath as one of the registerSubscriber arguments. The PAD will determine if the payload is well-formed XML and, if it is, it will apply this XPath expression on behalf of the subscribing application and will only send the portions of the payload that the application needs. The subscriber may also specify a notification listener that will be used by the Subscription Adapter to notify the application of information object arrival and loss. Finally, we allow the subscriber to specify both direct and indirect dissemination mechanisms. Specifically, the subscriber may choose to have the information object delivered directly into the application's address space (SA individual subscription queues) or he may choose to have the object delivered to a number of email recipients. The registerSubscriber method returns a unique identifier that identifies a specific subscription and is used in all subsequent SA API calls. This allows the subscribing application to register multiple

subscriptions using the same Subscription Adapter (in fact only one SA is allowed in a single address space).

The next two interactions are never seen by the subscribing application. In (6) the Subscriber Adapter receives a proxy to an appropriate PAD where the match has been made using equality based matching over the invariant portions of the information object metadata. In (7) the SA uses the PAD proxy to register the additional subscription information such as the XPath metadata predicate, XPath content-based filter, indirect email addresses, the subscription UID, and, most importantly, a proxy that the PAD may use to contact the Subscriber Adapter. The registered proxy is mostly used to push published objects into the address space of the subscribing application and to determine the health and status of the subscribing application.

We have mentioned earlier that the current implementation also supports a peer-to-peer implementation using the Publisher Adapter classes. The interactions in this case are essentially identical to the delegate discussion with the exception that we do not connectToDelegate and, when matched via the invariant portions of the information object metadata, the Subscriber Adapter is given a proxy to the Publisher Adapter that is running in the address space of the appropriate publishing application. The PA then handles all of the processing that we discussed earlier regarding PADs (i.e. publication queues, management of dissemination threads, metadata predicate processing, content-based filtering, email dissemination, etc.).

5. Exploitation of the Information space

Use scenarios and systems that are currently using the publish/subscribe capabilities are described in the following sections.

5.1. Airborne Experiment

The airborne experiment is a joint project involving the Air Force Research Laboratory (AFRL) Information Directorate's Intelligent Adaptive Communications Controller (IACC) and Joint BattleSpace Infosphere (JBI) teams.

The IACC project is focused on the requirements of Air Mobility Command (AMC) for global in-transit visibility and seamless, multi-media command and control. IACC provides dynamic network connectivity over multiple resources without control from a user. The in-house IACC team has provided information connectivity to airlifters and other remote/deployed users, by intelligently integrating and managing available communications resources. These resources include military communications such as HF and UHF SATCOM, as well as commercial resources such as Global Air Traffic Management (GATM) network, Inmarsat and Iridium. Through integration existing communications media, the system provides for cost effective (no aircraft mods)

information capability to deployed and in-transit assets. The IACC system integrates current, stove-piped communications systems to establish a secure, assured, real-time information and communications infrastructure.

The airborne experiment was developed to determine how the JBI Jini-Based Publication and Subscription Services would perform over extremely disadvantaged communications links. The IACC team maintains a testbed capability that allows for experimental transmission and characterization of traffic over IACC tactical links (i.e. UHF LOS, HF, and UHF SATCOM). During the first phase of the experiment the team instantiated an “airborne” client that used a metadata template to subscribe to an Air Tasking Order (ATO). An ATO can be rather large so, in order to decrease bandwidth utilization, a content-based filter (XPath expression) was provided at the time of subscription. The filter was applied by the dissemination node so that only mission information that pertained to the airborne node would be disseminated to the subscriber. Since the dissemination node (PAD) was ground based this significantly decreased the bandwidth utilized. During the first phase of the experiment the aggregate bandwidth ranged from 7200 to approximately 23000 bps. The first phase of the experiment proved that the Pub/Sub performed reasonably well over a disadvantaged communications substrate. In the future the team will explore how the Pub/Sub behaves while subjected to various communications degradations and failures. In addition, a study will be made of the control message exchanges spawned by the Pub/Sub infrastructure to determine whether optimizations may be made.

5.2. COABS Grid Integration

The Control of Agent-Based Systems [9] (CoABS) is a project that is funded by the Defense Advanced Research Projects Agency (DARPA) to develop and demonstrate techniques to safely control, coordinate and manage large systems of autonomous software agents. In general, the program has invested considerable resources in investigating the use of agent technology to improve military command, control, communication, and intelligence gathering.

The CoABS grid middleware is a service centric implementation that is built upon Jini. It was discovered that a Pub/Sub capability for the dissemination of information would be of great benefit to grid agent developers. Since this capability did not currently exist in the grid and a viable prototype was available in the JBI implementation, it was decided to integrate this capability within the grid. Currently the grid contains a full implementation of version 1.1.8 of the JBI Jini-Based Pub/Sub and is actively being used by a number of grid researchers.

5.3. Joint Experimentation

This publish/subscribe implementation is currently being used to disseminate information from several real and many simulated sensors. These sensors will stress the throughput of the pub/sub infrastructure both from a networking and an XML processing perspective. Simple fusion engines will subscribe to the published sensor ‘reports’ to construct fused (or correlated) outputs. Sensors that respond to tasking will also provide Jini services for this purpose.

6. Java 2 Enterprise Edition (J2EE)-Based JBI Reference Implementation

From the very beginning of the project, the Jini-based JBI RI core services to the current RI, one philosophy reigned supreme over all others: pluggability. The intention was that researchers who would like to integrate their innovative solutions to some of the underlying information management challenges could do so using the existing RI’s framework. In the following section, we will provide a brief walk through the architecture of the most J2EE recently released RI (version 1.2).

6.1. Client Application

Before discussing the inner workings of the implementation it is appropriate to spend some time describing how client applications view and interact with the information management infrastructure. The application designer views the JBI infrastructure as a collection of information management services that will be used to maintain, disseminate, and otherwise manage information objects as they are published by producers and used by consumers.

6.2. The Information Object

The managed information object (IO) is the fundamental construct dealt with by clients and the Infosphere. At its simplest, it may be thought of as a container which is comprised of *metadata* that describes a particular information object instance and a *payload* which may contain the IO specific information. In addition, the IOs are each organized by type. The structure of the metadata that describes the IO (its *schema*) is defined and registered within the infosphere’s Metadata Schema Repository (MSR) and associated with a type and version identifier. This subsystem provides a convenient set of interfaces that the client developers use when registering their types and that may be used when brokering for information. Since the published IOs are stamped with instance metadata that conforms to the structure and types stored in the MSR for that IO, a

consuming client may issue a subscription *predicate* or query *constraint* that would match their exact information requirements. This increases the likelihood that the client will receive only the information that is of interest when it becomes available.

6.3. The Common API (CAPI)

During the development of several JBI implementations, a group of collaborating researchers (infospherics.org) developed a collection of interfaces that client applications would use when interacting with the Infosphere known as the JBI Common Application Programming Interface (CAPI). The infospherics group [10] is an external group consisting of government, commercial, and academic stakeholders. In short, the CAPI provides interfaces that describe the constructs that must be implemented by any compliant JBI implementation. This approach would allow for client application portability among a number of disparate JBI implementations. Five fundamental constructs exist within the CAPI, namely, the Connection Manager, Connection, Publisher Sequence, Subscriber Sequence, and Query Sequence. A client application first creates a Connection Manager which acts as a Connection factory. The Connection Manager is then used to create a Connection. The Connection then may be used to authenticate with the Infosphere in order to allow only authorized interactions by the client. The Connection, if authorized to do so, may then create one of a number of Publisher, Subscriber, or Query Sequences. Sequences are created based on the type of information object that the client either wishes to produce or consume. Sequences may also have attributes associated with them that are provided as requests to the platform that may or may not be honored based on existing policy. This notion was included to accommodate tailoring behaviors associated with a sequence in the future. One example would be a level of Quality of Service (QoS) requested by the client application. Another may be a requested level of resilience or fault tolerance. Subscriber and Query sequences may also provide a *predicate* or *constraint* defined over the metadata to further refine the definition of what is to be delivered to the consuming client. In addition, Subscriber Sequences may be used to specify a client implemented callback that provides for asynchronous delivery of information objects based on type and, optionally, matching predicate.

In keeping with the spirit of pluggability mentioned earlier, the Reference Implementation v1.2 provides for a completely pluggable CAPI implementation (depicted in Figure 6). Currently the implementation class names (classes that implement the CAPI interfaces) are provided within a properties file. A platform developer need only provide the names of his CAPI implementation classes in order to allow our client runtime to interact with his platform.

The RI currently allows for two modes of underlying interaction with the platform. One mode allows interaction via a Web Services [11] layer using SOAP [12] messages. The second allows direct interaction with a collection of J2EE session beans. We used the pluggability within the client-side implementation to choose between one mode and the other. In both cases the client uses the CAPI and is not aware of the underlying techniques and wire protocols that are being used.

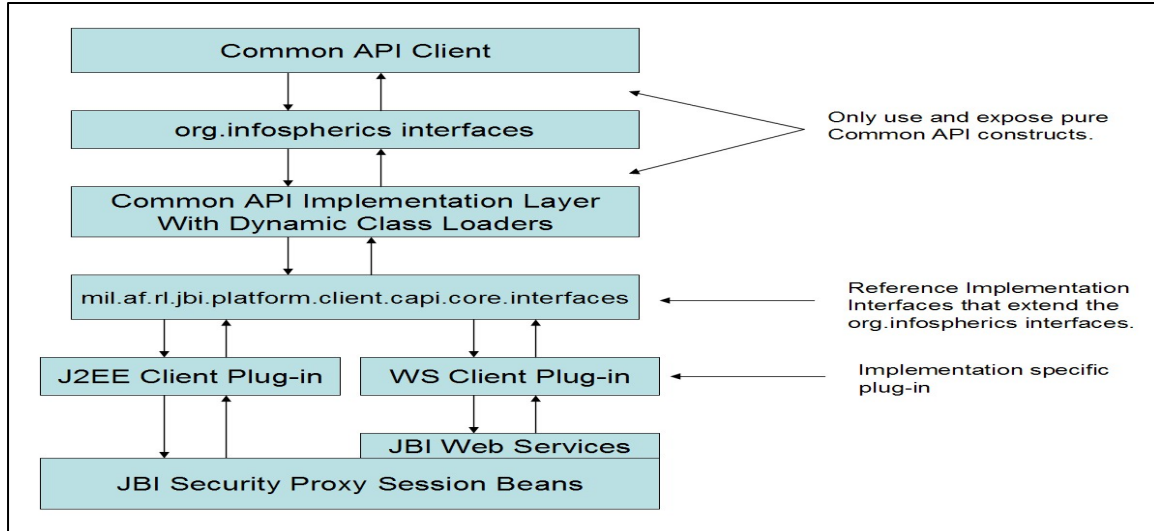


Figure 6: The Pluggable Common API

6.4. Version 1.2 Architectural Design

In this section we will provide a short description of the major services provided by the underlying information management infrastructure. A depiction of the RI version 1.2 architecture is provided. While this implementation could certainly be ported using a number of implementation technologies, we selected the Java 2 Enterprise Edition (J2EE) standard [13] for our current implementation. We selected JBoss [14] as an exemplar implementation of the J2EE standard. The JBoss application server is a robust and freely available implementation that is offered with source code. This makes it particularly attractive as a research vehicle.

We mentioned in the previous section that client applications may interact with the core services via two different modes. The first layer depicted in Figure 7 represents the server-side Web Services layer. The constructs available within the CAPI are implemented as a collection of services within the web tier that allow for interaction using the standard SOAP messages. The Web Services layer on the server-side is essentially nothing more than an alternate point of entry. It accepts SSL encrypted SOAP messages and processes the messages so that the appropriate Java object web service implementation is invoked with the proper parameters. The request is then handed off to the appropriate J2EE Security Proxy Stateless session layer. If the, more optimized, J2EE session mode of interaction is used by the client then the pluggable client implementation classes interact directly with the Security Proxy stateless session layer.

The Security Proxy Layer is used as the security gate through which all authorization checks are made. For example, as the client attempts to publish, subscribe, or query for a specific information object type, the security proxy sessions interact with the JBI Gatekeeper service in order to verify that the authenticated user currently has the sufficient level of privilege for the requested action. It is this role-based access control implementation that mediates all access to information and actions performed within the information management infrastructure.

Once the client interaction has been authorized, the request is passed to the generic implementation layer. This layer provides minimal additional processing and, for the most part, translates requests into concrete invocations on lower level implementation services. For example, if the request is from a publisher or subscriber requesting sequence activation then this would cause an interaction with components that provide for information object dissemination.

The InfoObject Dissemination Services or “Engine” has the primary role of providing a pluggable capability for predicate evaluation, subscriber queue management, and object proliferation or dissemination. Researchers may specify their own predicate evaluation implementations and drop them into the RI deployment. This allows for ultimate flexibility as we explore different constraint languages and metadata representations. The implementation supports XPath [15] as a default constraint or predicate language. The predicate definition is an XML document specifying a predicate name, predicate type, and the actual predicate. The client developer need only specify a predicate type that matches a particular predicate evaluator that is supported and the predicate will be associated and used for the specific subscription.

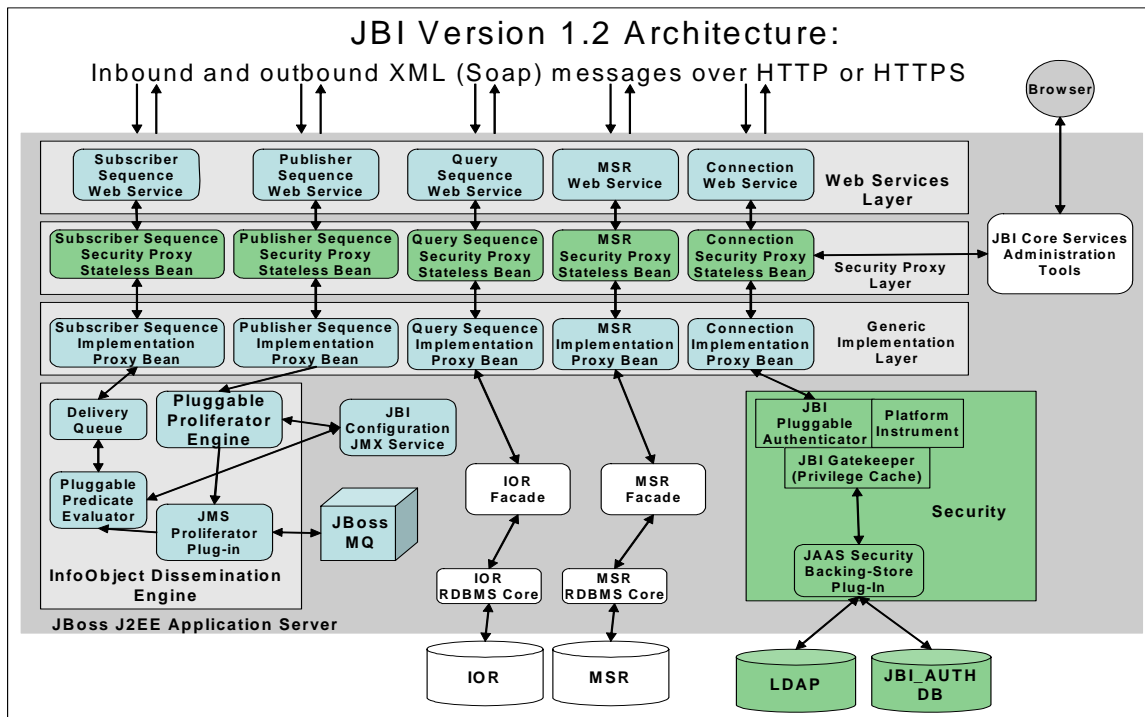


Figure 7: JBI Core Services

In addition, as a first attempt at providing a more flexible way of adding new wire protocols and dissemination technologies, we have introduced a *proliferator* and *receptor* model. The intent is that a proliferator and receptor be introduced and function as a pair. Normally the receptor would live within the consuming client's address space, while the proliferators, not surprisingly, would function within information management services layer (specifically within the InfoObject Dissemination Services). As an example, one of the default pairs that are used within the current version uses JMQ (a JBoss JMS implementation) to disseminate information objects to subscribing client consumers. A particular receptor may be specified within the aforementioned client properties file while matching proliferators may be developed and specified through the JBoss service console.

When an information object is published, its instance metadata is used in making a dissemination decision. The list of subscriptions associated with information objects of that type is formed and the appropriate pluggable predicate evaluator is used to produce the dissemination decision. At this point one of the dissemination services interacts with the JBI Gatekeeper to determine if the subscriber has the necessary privileges to receive the information object. If the object is to be disseminated, it is placed on the subscriber's queue. As the information object is pulled from the queue by the dissemination engine, the appropriate pluggable proliferator is used to push the object into the client subscriber's address space. It is at this point that the client's local callback is activated and the information object may be consumed.

When a query is issued via a Query Sequence, it is ultimately forwarded to the Information Object Repository (IOR) Façade Bean. This bean is located and used based on whatever underlying RDBMS implementation the substrate is using. In version 1.2, we support both MySQL and Oracle as underlying implementations where the XML metadata schema is mapped into relational tables in order for information objects to be retrieved based on selected the metadata elements. These data stores require a restart of the service when switching from one data store to another. In the future, this capability will be replaced by a JMX service (MBean), which allows for the hot data store swapping. In addition, future implementations will support XML database capability.

The query predicates that have been defined in XPATH are converted to SQL for execution. Invocations are then forwarded to the appropriate underlying IOR core components that have the responsibility of issuing the appropriate SQL queries to the underlying database and returning information object result sets to the façade layer for delivery to client. It should also be mentioned that the Metadata Schema Repository (MSR) has client (CAPI) interfaces that allow for the specification of information object types and the metadata schemas associated with that type. Like all other interactions with the Infosphere, clients must have authorization to interact with the MSR. The underlying MSR Façade and Core component implementations mimic the IOR implementation closely.

The Security Services provide a JBI Pluggable Authenticator capability and a JBI Gatekeeper service. The Gatekeeper is the fundamental underlying service used in performing authorization checks and, in addition, provides for user privilege caching. The default authenticator uses a Java Authentication and Authorization Service (JAAS) [16] compliant backing store to maintain user privileges. When a specific Connection is authenticated the JBI Gatekeeper caches the privilege information obtained from the JAAS backing store and maintains it based on the connection UID. This approach is used in order to improve the efficiency of authorization checks.

A set of web based Information Management Staff (IMS) tools have also been developed, thereby easing the burden of administering the JBI. This set of interfaces allows for the creation of users (principals), association, and revocation of specific privileges with users, information object type and metadata schema definition, and information object repository management (object instance removal).

7. Future Directions

In this section, we will describe some of the new features and enhancements that will be coming in the next release of the JBI reference implementation and in future releases.

7.1. Discovery

The next release of the information management platform services will support two types of discovery. The client-side CAPI implementation classes are currently pluggable and dynamically loaded from libraries that are made available on the client's machine. In the next release, we will use a Universal Description, Discovery and Integration (UDDI) [17] server in order to discover an appropriate class server. This web service will be used to provide for the appropriate client-side class implementations based on information provided in the *connection descriptor*. The *connection descriptor* is currently provided when creating a new connection using the CAPI's ConnectionManager. It is currently underutilized but it will minimally identify the platform of interest by name and/or type. The class server will use this information in order to provide implementation classes that are compatible with a particular desired JBI platform implementation.

The UDDI server will also be used to obtain endpoint information that is necessary when directing invocations to a specific JBI service instance. This endpoint information would likely be useful regardless of the client and server-side implementation technologies that are being used. For example, in the current RI implementation, we provide two pluggable client-side CAPI implementations. One implementation uses Web Services, while the other interacts directly with J2EE session beans. In both cases, some endpoint information is necessary. In the Web Services case, a URI is necessary to direct invocations to the appropriate server-side Web Services CAPI implementation classes. For the J2EE case, a Java Naming and Directory Interface (JNDI) [18] reference (a protocol specific URI) is used as the endpoint of interest. This will enable the discovered and dynamically loaded client-side CAPI implementation classes to interact seamlessly with appropriate information management services. We will further expand the notions of discovery to be of more generic use by implementing Infosphere system service components. This will allow for the more flexible use of components within the information management infrastructure.

In the future, we will be researching how mechanisms may be used in the discovery of *information* rather than implementing services. This is thought to be a longer term research project that may have implications on how we characterize and manage information within the existing Infosphere. As such, it probably will not appear in the next release of the reference implementation.

7.2. Policy Representation and Enforcement

The existing JBI implementation provides for a robust role based access control capability that allows for fairly fine-grained control over access to the Infosphere and dissemination of information. Within the RI, information object type names are specified within packages, similar to Java language implementations. Information management staff personnel may specify whether an authenticated user may Publish, Subscribe, Query, and/or Archive information objects of a specific type. In addition, wildcarding is allowed so that privileges may be defined for a package and all sub packages and types

within the hierarchy. The language and mechanisms used to represent and enforce these access policies are inadequate and do not conform to a standard.

We are currently experimenting with different approaches for policy representation and enforcement within the Infosphere. As a first candidate, we are prototyping a role based access control capability using the eXtensible Access Control Markup Language (XACML) [19]. XACML is entirely XML based and was originally designed for access control. The language has been standardized by the OASIS (Organization for the Advancement of Structured Information Standards) group [20]. XACML embodies both an access control policy language (basically designed to describe who may do what and when) and a request-response based language that allows users to provide queries to determine whether or not a particular access should be allowed.

Typically a *subject* (principal or user in our parlance) may want to perform some *action* on a particular *resource*. The subject would then issue a query on a service or component that has the responsibility for protecting the resource (in our case, information). This service is commonly referred to as the Policy Enforcement Point (PEP). The PEP service would then create a well formed and valid request based on the subject, action, and resource using the XACML request language. The PEP service would then forward the request to the Policy Decision Point (PDP) service or component which has the responsibility for evaluating the requests using applicable XACML policies in order to determine whether access should ultimately be granted. The response would then be returned back to the PEP service which either allows or denies access as appropriate.

XACML was selected as an initial target language for a number of reasons. It provides us with one standard access control policy language that would replace our rather limited representation. We envision using XACML in a more expansive role to represent and enforce policy throughout the information management system (i.e. QoS and resource management). We anticipate that the quantity and quality of tools for writing and managing XACML policies will be developed over time which will further alleviate some of the administrative burden placed on information management staff personnel. We have found that the XAML policy language is more than adequate to represent all of the existing information access control policies that we have currently defined in our reference implementation. XACML will easily support access control policy specifications that are not currently available in the reference implementation (negative policy, etc.). The language is extensible and would seem to be able to accommodate all of the policy specifications that we may require in the future.

In addition, as part of an in-house research project, we are exploring how some of the more recently available semantic web languages may be used to represent and enforce policy within an information management system. Semantic web languages (DAML, OWL) have been researched with respect to policy representation in large, complex distributed systems [21]. The expressive capabilities of semantic web languages make them applicable to many domains. The available tools for manipulating, visualizing, and reasoning over these languages also provides significant utility. One such application of semantic web languages to policy representation, reasoning, and enforcement is the

Knowledgeable Agent-oriented System (KAoS) [22]. KAoS uses an ontological representation of policy and provides services that enable for the application, reasoning, and enforcement of these policies. KAoS also provides a powerful user interface KPAT (KAoS Policy Administration Tool) that simplifies the authoring and manipulation of these policies. We are planning to explore how KAoS may be used within the JBI reference implementation in order to better describe policy based on the relationships that are defined between different types of information.

7.3. Instrumentation and Control

The JBI Instrumentation Services give users insight into what activity is occurring inside the reference implementation. The initial Instrumentation Services implementation was developed as a standalone system that interacted with the RI through a set of low impact interfaces. [23] This initial Instrumentation Services Architecture made use of the Instrumentation Entity Model to create entities that describe client objects interacting in the RI: platforms, connections, users, nodes, and sequences. These instrumentation entities (implemented with J2EE Entity Beans) populate the Instrumentation Space (a MySQL database) and are accessed by clients through the Instrumentation Client API (ICAPI). A web-based client that made use of this ICAPI was developed to visualize instrumentation information and demonstrate the capabilities of the Instrumentation Services. This client utilizes numerical rate graphing (using JFreeChart) and a dynamic graph tree (using TouchGraph) to visualize JBI activity, as seen in Figure 8.

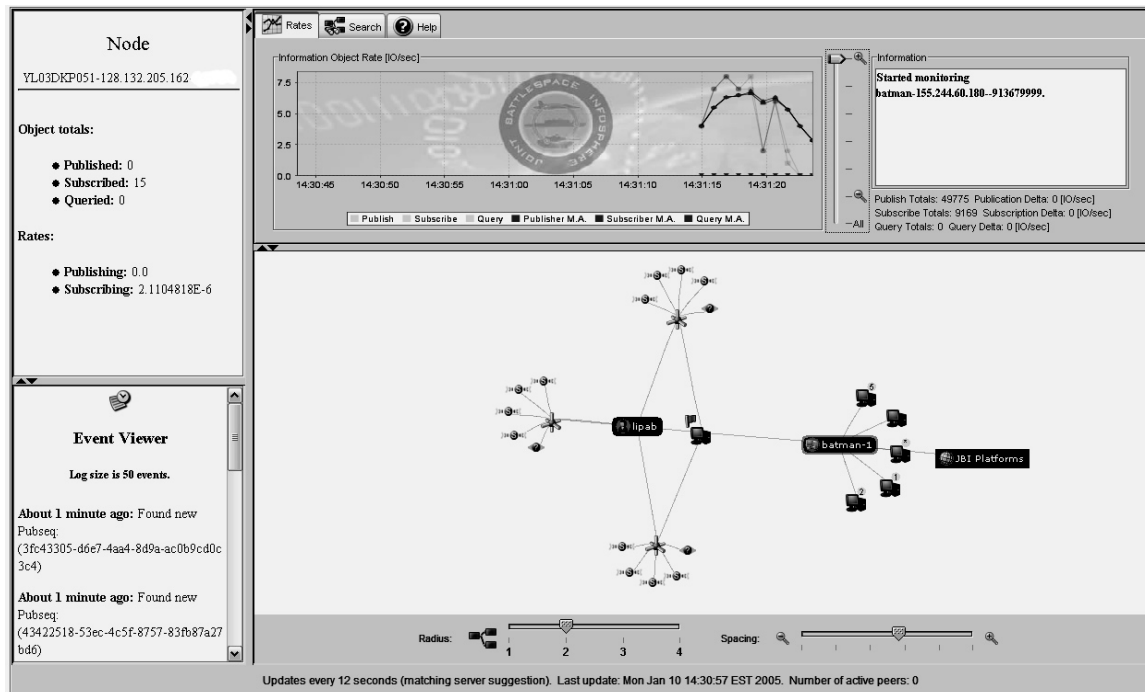


Figure 8: JBI Instrumentation Visualization Client

Current JBI Instrumentation Services development is focused on the Reference Implementation version 1.2.5 requirements. To make the Instrumentation Services Architecture more efficient, we have decided to move the Instrumentation Services into the RI. The RI now directly records instrumented events and information using JBoss Cache technology. Communication with instrumentation clients is still carried out through the ICAPI, which, in version 1.2.5, wraps a local copy of the instrumentation cache maintained on the client by asynchronous messages transmitted from the RI. Added functionality to the Instrumentation Services will allow clients to display MIO traffic patterns and derive actionable statistics about MIO sources and destinations.

In addition to instrumentation, future ICAPI versions will provide instrumentation clients with control capabilities. Such control may include the ability to terminate connections, throttle back the MIO throughput of a sequence, or graphically modify user privileges. Combined with already existing instrumentation capabilities, this control functionality will allow clients to execute policy by monitoring instrumentation information, making policy decisions, and then effecting control over the RI. A client with this control capability would be an indispensable tool for the Information Management Staff. Current research into policy languages and engines will fuel ongoing instrumentation development.

7.4. *Information Transformation*

The transformation services will enhance the value of the information disseminated by the RI. The services will manage the storage, maintenance, and lifecycle of manipulation mechanisms known as fuselets. A fuselet is a lightweight, special-purpose client program that provides value-added information processing functions over the managed information object that is under the control of the RI. The fuselet's information processing functions manipulates incoming information objects in specific ways (defined by the fuselet logic) to produce new information objects for publication. In short, fuselets enable information to be manipulated into a form that is required by and useful to the warfighter.

The objective of fuselet technology is to enhance information systems by providing a flexible information production capability which may adapt to the changing needs of end users while requiring little or no change to legacy client applications. A fuselet runtime capability will have the effect of improving the efficiency and effectiveness of decision-making by correlating duplicative information, resolving inconsistent information, mediating between information sources, and fusing information together into comprehensible information products.

In order for fuselets to be useful their lifecycle must be carefully managed. The difficult task of developing of a runtime environment that manages a variety of fuselets implemented in multiple languages (XSLT, Jython, Java, Jess, Groovy, etc.), operate on a variety of information object types and payload formats, operate in multiple contexts (e.g., COI ontologies/semantics), and possess different execution properties (e.g., stateful

versus stateless) is still underdevelopment. While the complete solution has yet been realized, AFRL has developed a prototype that establishes a basis for a fuselet execution and management environment [24]. The fuselet runtime environment (FRE) and fuselet runtime management environment (FRME) are currently being developed and will be deployed a future release of the reference implementation.

7.5. *JB1 and Net-Centric Synergy*

A key challenge for the DoD is to “improve the speed and quality of [warfighter] decision making by connecting information producers and consumers more effectively through information technology and net-centricity.” The JB1 project is experimenting with advanced information management technologies and applying those concepts in the formulation and evolution of an effectively managed information space. We anticipate a variety of domain-specific information spaces being identified to support groups of users exchanging information in pursuit of shared goals or missions. These groups of users are referred to as a Communities of Interest (COI).

7.6. *Community of Interest*

COI [25] is a term used to describe any collaborative group of users who must exchange information in pursuit of their shared goals, interests, missions, or business processes, and who therefore must have shared vocabulary for the information they exchange. The COI concept is intended to be broad, and cover an enormous number of potential groups of every kind and size. In the creation of a COI there are several tasks that must be completed before an effective net-centric sharing process can be achieved. First, a detailed information engineering process must be performed to develop the information object types and metadata schemas that will be used as the vocabulary to provide the semantic and syntactic understanding within the share information space. The metadata schema should then be registered in the DoD Metadata Registry for visibility and reuse. This will allow other communities and users throughout the DoD to use related constructs to exchange information. Reusing existing metadata will tend to require less mediation or transformation when information is exchanged between COI's.

7.7. *Air Force C2 Constellation*

The C2 Constellation [26] program has been created to assist the Air Force in building a network centric, peer based and system of systems, which operate in a seamless and fully interoperable framework. The Air Force vision is a connected array of land, platform, and spaced based sensors that use common standards and communication protocols to relay information automatically in what he refers to as “machine to machine interface.” This effort to integrate systems is referred to as enterprise integration. The C2 Constellation is tackling the networking solutions and connectivity issues in a two-tiered approach. At the

top tier the C2 Constellation seeks to define C4ISR enterprise integration. At the bottom tier the C2 Constellation solves near term, quick turnaround integration solutions for the Air Force. The JBI research team is evaluating the C2 architecture and how the JBI IM services can support edge user development and integration. The current COI application communities within the C4ISR enterprise being targeted by the JBI research team are the intelligence and tactical communities, with additional feasibility demonstrations in the C2 community.

7.8. Global Information Grid Enterprise Services

Several initiatives have been undertaken to establish net-centric capabilities across the DoD. The Global Information Grid (GIG) [27] will be a net-centric system operating in a global context. The system will provide processing, storage, management, and transport of information. The GIG will support the Department of Defense (DoD), national security agencies, and related Intelligence Community missions and functions across strategic, operational, tactical, and business-in war, and will do so in crisis, as well as peace. The GIG will provide the foundation for net-centric operations by globally interconnecting the building blocks of information capabilities, including: the physical communication links, hardware, software, data, personnel and processes. The GIG Enterprise Services (ES) is an umbrella term to describe the information services that reside on the GIG. The Network Centric Enterprise Services (NCES) is a DISA-sponsored program that seeks to provide a generic set of core services that will be available to all DoD edge users allowing them to “pull mission-tailored information intelligently from anywhere within the network environment [28].”

7.9. Net-Centric Enterprise Services

The initial NCES core services are being developed using the latest industry standards, such as extensible markup language (XML) and web services, and will provide functionality as services to GIG end-user applications. This architectural approach and its use of standards and web technologies are referred to as a Service-Oriented Architecture (SOA). SOA and other enterprise architecture approaches are based on a collection of independent and networked business components where all interactions are achieved through well-defined, published, and standards-compliant interfaces. The SOA concept defines three levels of abstraction: Operations, Services, and Business Processes. Operations represent single units of work, and when logically grouped, form a Service. The table below illustrates an example of a service called “Inventory Control” consisting of several operations related to inventory items.

<i>Service</i>	<i>Operations</i>
<i>Inventory Control</i>	Lookup items by reference number
	List resources by name and region
	Save data for new items

Figure 9: An Example of a Service

A business process consists of a long running set of actions or activities performed with specific business goals in mind, and when layered on top of work order processing, would involve activities such as “Sell Products” or “Fulfill Orders”. Business processes typically encompass multiple service invocations either one at a time or several at a time in an orchestrated fashion [29]. Examples of USAF business processes could be activities conducted in support of an air campaign, such as Air Tasking Order (ATO) planning or Target Nomination List (TNL) generation.

7.10. JBI and GIG Enterprise Services

Within the GIG infrastructure, multiple JBI-enabled COIs and other edge user applications will be established to satisfy specific end user needs. These JBI-enabled COIs will be capable of sharing their information among each other as well as other edge users operating on the GIG. The net result is that each user will have capabilities above and beyond what any individual service capability provides.

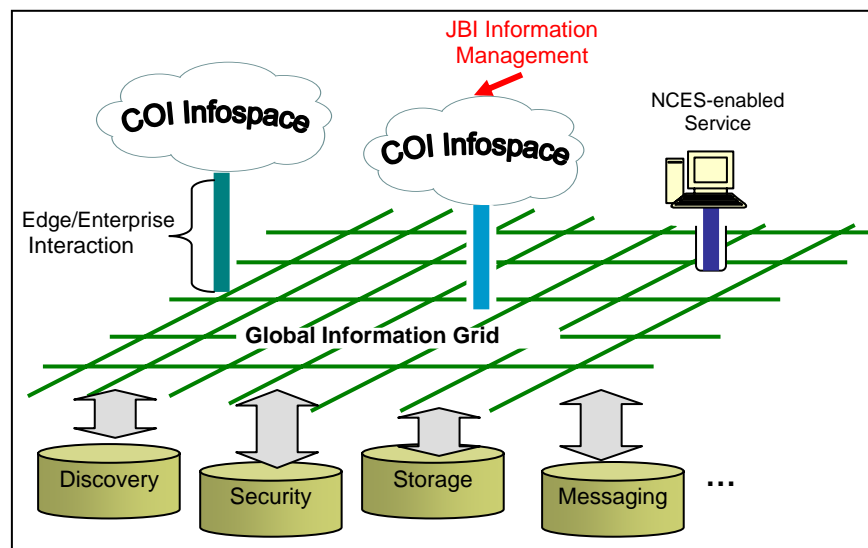


Figure 10: JBI enabled Information space facilitate edge user interaction in conjunction with GIG enterprise services.

Figure 10 illustrates this overall concept of NCES- and JBI-enabled COIs operating in conjunction with other edge user applications existing on top of the underlying GIG foundation. In reality, the GIG infrastructure will provide the DoD enterprise transparent

and seamless access between the NCES core services, JBI information spaces and edge user services. On their own, JBI Information Management (IM) services provide information management and exchange capabilities that support tailored, dynamic, and timely access to required information to enable near real-time planning, control, and execution for DoD decision making. However, JBI IM services will be capable of integrating with the NCES core services and when deployed, will establish an interoperable “information space” that aggregates, integrates, fuses, and intelligently disseminates relevant information to support effective warfighter business processes. This virtual information space provides individual users with information tailored to their specific functional responsibilities and provides a highly tailored repository of, or access to, information that is designed to support a specific COI, geographic area or mission. This information space also serves as a clearinghouse and a workspace for anyone contributing to the accomplishment of the operation—for example, weather, intelligence, logistics, or other support personnel.

In combination with NCES core services, JBI IM services will be utilized to dynamically create an information space and manage edge user interaction within a COI. An example of this, from a GIG ES perspective, would allow edge users to discover relevant COIs (using NCES discover services), gain access to the information space (JBI access controls using NCES user profiles and underlying security mechanisms), and exchange or manipulate relevant information (using JBI IM services). In this manner, a JBI-enabled COI would have the full suite of JBI information management capabilities, but would still be based on and have full access to NCES services. Together, NCES and JBI IM services will be utilized to develop end user applications targeted at mission-specific COIs to facilitate a secure, controlled information exchange and decision support environment.

7.11. JBI and GIG Net-Centric Services

The JBI research team has established a local NCES testbed, a Solaris server which hosts the current NCES development tree and available services. The intent is to host the emerging NCES services as they are being developing and evaluate their application. The goal is to identify JBI, GIG-ES, and C2 Constellation gaps and intersections. The JBI research team will take the opportunity to influence potential areas where JBI provides a capability that does not exist within the GIG ES. It is critical that the available services are evaluated to identify which ones can be directly used, how the JBI IM services can be integrated within the GIG ES and what potential technical hurdles must be overcome to provide a cohesive foundation for edge user development and integration.

Figure 11 below illustrates an initial evaluation that was conducted by one of the JBI research contractors as a starting point in assessing potential JBI and GIG ES synergy. The results of this analysis will be presented in the future technical report.

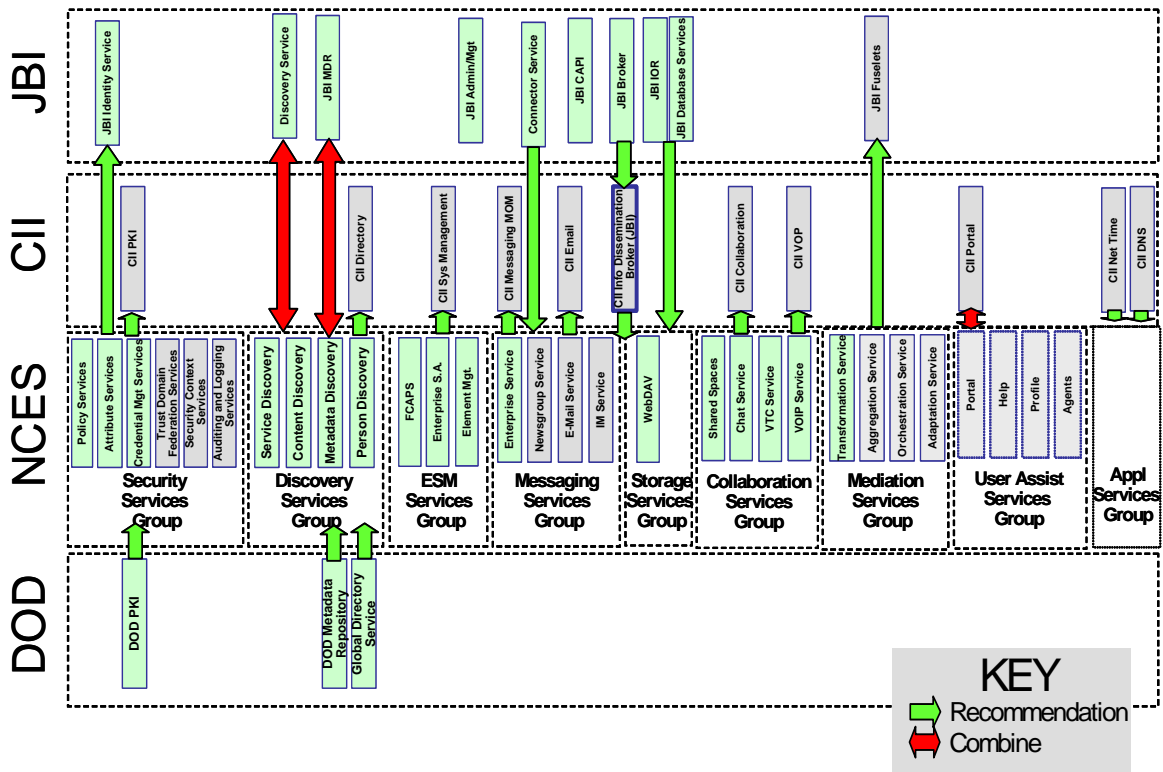


Figure 11: Initial Evaluation of GIG ES intersection

8. Conclusion

Current research and development under the JBI information management technology area is focused on a vision of effectively building and managing an information space. The JBI Common API provides an isolation layer between client applications and the underlying technology. This technology isolation allows this reference implementation or any implementation developed to be based on the latest commercial technologies and therefore insure that future innovation can continue as the commercial IT technology base moves forward. The JBI enabled information space is intended to support virtually any operational domain; short term utilization such as humanitarian relief to larger long-term COIs supporting edge users interacting within the GIG. Future work will address the information engineering process to support domain-specific COI interactions including internal legacy systems as well as the transformation requirements to enable interoperability among various COIs residing on the GIG.

9. REFERENCES

- [1] USAF Scientific Advisory Board, “Information Management to Support the Warrior”, SAB-TR-98-02, <http://www.rl.af.mil/programs/jbi/documents/IMReport.pdf>, 1998.
- [2] USAF Scientific Advisory Board, “Technology Options to Leverage Aerospace Power in Operations Other Than Conventional War”, SAB-TR-99-01, http://www.rl.af.mil/programs/jbi/documents/TLAP_Final_Volume_1.pdf, 2000.
- [3] USAF Scientific Advisory Board, “Building the Joint Battlespace Infoshere Volume 1: Summary” SAB-TR-99-02, <http://www.rl.af.mil/programs/jbi/documents/JBIVolume1.pdf>, 1999.
- [4] Java Message Service Documentation, “Java Message Service Specification Version 1.0.2b”, <http://java.sun.com/products/jms/docs.html>, 2001.
- [5] World Wide Web Consortium, “Extensible Markup Language (XML) 1.0 (Second Edition)”, <http://www.w3.org/TR/2000/REC-xml-20001006>, 2000.
- [6] World Wide Web Consortium, “XML Schema Part 0: Primer”, <http://www.w3.org/TR/2001/REC-xmlschema-0-20010502>, 2001.
- [7] Sun Microsystems Inc., “Jini Architecture Specification”, http://www.sun.com/software/jini/specs/jini1_2.pdf, 2001.
- [8] J. A. Moore, C. Salisbury, “Reconfigurable Simulation Visualizer”, *Proc. SPIE* **4026**, pp. 50-54, 2000.
- [9] Kahn, Martha and Della Torre Cicese, Cindy. “The CoABS Grid.” In Goddard/JPL Workshop on Radical Agent Concepts (WRAC), 2001.
- [10] The Infospherics Group, <http://www.infospherics.org>
- [11] W3C, “Web Services,” <http://www.w3.org/2002/ws/>, 2002.
- [12] W3C, “SOAP,” <http://www.w3.org/TR/soap/>, 2003.
- [13] Sun Microsystems, “J2EE Version 1.3 Specification,” http://java.sun.com/j2ee/j2ee-1_3-fr-spec.pdf, 2001.
- [14] JBoss, “JBoss Application Server,” <http://www.jboss.org/products/jbossas>, 2005.
- [15] W3C, “XPath,” <http://www.w3.org/TR/2005/WD-xpath20-20050211/>, 2005.

- [16] Sun Microsystems, “Java Authentication and Authorization Service (JAAS),” <http://java.sun.com/products/jaas/> , 2004.
- [17] OASIS, “UDDI Specification,” <http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv3> , 2004.
- [18] Sun Microsystems, “Java Naming and Directory Interface (JNDI),” <http://java.sun.com/products/jndi/> , 1999.
- [19] OASIS, “eXtensible Access Control Markup Language (XACML),” http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml , 2005.
- [20] OASIS, <http://www.oasis-open.org/home/index.php> , 1993.
- [21] G. Tonti et al., “Semantic Web Languages for Policy Representation and Reasoning: A Comparison of KAoS, Rei, and Ponder,” The Semantic Web-ISWC 2003: 2nd International Semantic Web Conference, LNCS 2870, Springer-Verlag, 2003, pp. 419-437.
- [22] Andrzej Uszok et al., “KAoS Policy Management for Semantic Web Services,” IEEE Intelligent Systems, July 2004, pp. 32-41.
- [23] M. T. Muccio et al., “JBI health and status instrumentation services,” SPIE Defense & Security Symposium, March 2005
- [24] N. O. Ahmed et al., “Fuselets: lightweight applications for information manipulation,” SPIE Defense & Security Symposium, March 2005
- [25] The Department of Defense, Chief Information Officer, Information Management Directorate, “Communities of Interest in the Net-Centric DoD Frequently Asked Questions (FAQ),” May 19, 2004
- [26] Col Normal Sweet et al., “The C2 Constellation A US Air Force Network Centric Warfare Program”, CRRTS, June 2004
- [27] Global Information Grid, <http://www.nsa.gov/ia/industry/gig.cfm>
- [28] Department of Defense web page, Global Information Grid Enterprise Services The Challenge, <https://ges.dod.mil/about/challenge.htm> , August 20, 2004
- [29] Olaf Zimmermann, Pal Krogdahl, Clive Gee, “Elements of Service-Oriented Analysis and Design” <http://www-106.ibm.com/developerworks/webservices/library/ws-soad1/> , August 20, 2004.